多 MapReduce 作业协同下的大数据挖掘类算法资源效率优化 *

廖 彬[1†]，张 陶[2]，于 炯[2]，黄静莱[1]，国冰磊[2]，刘 炎[3]

(1. 新疆财经大学 统计与信息学院，乌鲁木齐 830012; 2. 新疆大学 信息科学与工程学院，乌鲁木齐 830008; 3. 清华大学 软件学院，北京 100084)

摘 要：由于任意的 MapReduce 作业都需要独立的进行任务调度、资源分配等一系列复杂的操作，这使得同一算法协同的多个 MapReduce 作业之间，存在着大量的冗余磁盘 I/O 及资源重复申请操作，导致计算过程中资源利用效率低下。大数据挖掘类算法通常被切分成多个 MapReduce Job 协作完成，以 ItemBased 算法为例，对多 MapReduce 作业协同下的大数据挖掘算法存在的资源效率问题进行了分析，提出基于 DistributedCache 的 ItemBased 算法，利用 DistributedCache 将多个 MapReduce Job 之间的 I/O 数据进行缓存处理，打破作业之间独立性的缺陷，减少 Map 与 Reduce 任务之间的等待时延。实验结果表明，DistributedCache 能够提高 MapReduce 作业的数据读取速度，利用 DistributedCache 重构后的算法极大地减少了 Map 与 Reduce 任务之间的等待时延，资源效率提高 3 倍以上。

关键词：MapReduce 优化；ItemBased 算法；内存文件系统；I/O 效率；资源优化

中图分类号：TP393.09 doi: 10.19734/j.issn.1001-3695.2018.11.0795

Resource efficiency optimization for big data mining algorithm with multi MapReduce collaboration scenario

Liao Bin[1†], Zhang Tao[2], Yu Jiong[2], Huang Jinglai[1], Guo Binglei[2], Liu Yan[3]

(1. *College of Statistics & Information*, *Xinjiang University of Finance & Economics*, *Urumqi* 830012, *China*; 2. *School of Information Science & Engineering*, *Xinjiang University*, *Urumqi* 830008, *China*; 3. *School of Software*, *Tsinghua University*, *Beijing* 100084, *China*)

Abstract: Because any MapReduce job requires a series of complex operations such as task scheduling and resource allocation independently, there are a lot of redundant disk I/O and resource duplicate application operations among multiple MapReduce jobs coordinated by the same algorithm, causing inefficient resource utilization in job computing process. Big data mining algorithms are usually divided into several MapReduce Jobs, taking ItemBased algorithm as an example, this papere analyze the resource efficiency of mining algorithm with multi-MapReduce job collaboration scenario. It proposed an ItemBased algorithm based on DistributedCache, which used DistributedCache to cache I/O data between multiple MapReduce Jobs, breaks the defect of independence between jobs, and reduced the waiting delay between Map and Reduce tasks. The experimental results show that, DistributedCache can improve the data reading speed of MapReduce jobs. The algorithm reconstructed by DistributedCache greatly reduces the waiting delay between Map and Reduce tasks, and improves the resource efficiency by more than three times.

Key words: MapReduce optimization; ItemBased algorithm; memory file system; I/O efficiency; resource optimization

0 引言

据 IDC(Internet Data Center)2015 年发布的报告[1]显示，全世界在 2015 年产生的数据总和接近 10 ZB，并且这一数字预计到 2020 年将达到 44 ZB。数据爆发式的增长为 IT 产业带来了机遇与挑战，数据产生的过程由被动模式转变为主动产生，标志着大数据时代的来临。大数据的规模效应导致其存储、计算、分析及管理等成本不断上升，这使得高效率低成本的大数据计算技术逐渐成为学术界及工业界的研究热点。自谷歌 2003 年发表论文公开分布式存储系统 GFS[2]及计算模型 MapReduce[3]以来，MapReduce 计算模型逐渐成为 Hadoop、Spark、Pig、Hbase 及 Hive 等大数据系统最通用的底层计算框架。

MapReduce 与之前的并行计算模型(如 PRAM、MPI 等)相比，根本区别是 MapReduce 秉承“移动计算”而非“移动数据”的思想，“移动计算”的核心本质是将计算程序调度到离数据最近(本地)的计算节点，以达到减少数据密集型作业计算过程中的数据传输量的目的。但是当 MapReduce 作业(Job)被调度系统分解成若干任务(Task)后，由于任务之间并不是孤立存在的，任务之间的协同执行过程，需要大量的磁盘读写和网络的传输操作(MapReduce 任务执行流程如图 1 所示)，比如 Split、RecordReader、Partition 及 Shuffler 等操作，都会涉及将中间计算结果在不同机器之间网络传输并存到磁盘上作为之后 pipeline 输入的操作。特别基于 MapReduce 模型实现的大数据挖掘类算法，由于复杂度较高，算法通常需由多个 MapReduce 作业协作完成，如 MapReduce 下的 PageRank

收稿日期：2018-11-08；**修回日期：**2019-01-03 **基金项目：**新疆维吾尔自治区自然科学基金资助项目（2016D01B014）

作者简介：廖彬（1986-），男（通信作者），副教授，硕导，博士，主要研究方向为绿色计算、数据挖掘、大数据计算模型等（liaobin665@163.com）；张陶（1988-），女，博士研究生，主要研究方向为分布式计算、网格计算；国冰磊，女，博士研究生，主要研究方向为绿色计算、数据库系统等；于炯（1964-），男，教授，博导，博士，主要研究方向为网络安全、网格与分布式计算；黄静莱（1996-），女，硕士研究生，主要研究方向为大数据计算；刘炎（1990-），男，硕士研究生，主要研究方向为大数据计算.

算法会被分解为 4 个作业，而最常见的 Bayes 算法则会被分解为 10 个作业。但是同一算法下的多个 MapReduce 作业之间，资源的调度与管理并不协同，严重的冗余磁盘读写及重复的 I/O 资源申请操作，造成算法的资源利用效率较低[4~6]。

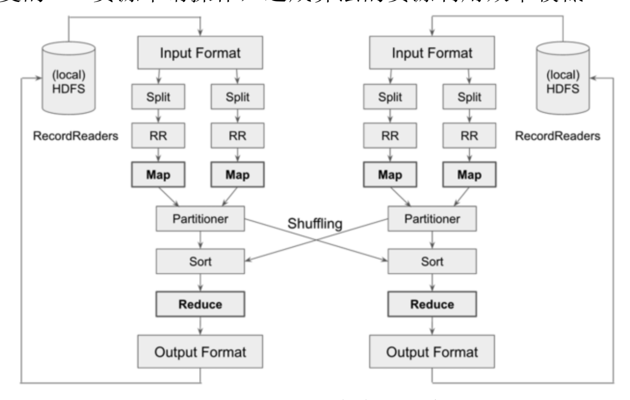


图 1　MapReduce 任务执行流程

Fig. 1　MapReduce's task execution process

推荐算法属于大数据挖掘类算法中应用最广泛的一种，本文以 ItemBased 协同过滤算法为研究对象，研究大数据挖掘类算法多 MapReduce 作业之间的资源效率问题，本文做了如下几个方面的工作：首先，对 MapReduce 环境下的 ItemBased 算法存在的资源效率问题进行了分析；其次，在问题分析的基础上，提出将同一算法协同的多个 MapReduce 作业之间的 I/O 数据进行统一的缓存处理，整体上减少算法计算过程中冗余的 I/O 操作，从而达到优化资源利用效率的目的；最后，通过对比实验验证了资源利用效率的提升。

1　相关研究

针对大数据挖掘类算法的研究，以推荐算法为例，大部分研究目标都是以提高算法在特定应用场景下(如电子商务、新闻推荐、视频推荐等)的推荐质量。评价算法优越性的主要参数有准确率(Precision)、召回率(Recall)、F 值(F-Measure)、E 值(E-Measure)、平均正确率(Average Precision)等。很少研究会关注到算法的执行环境优化、资源利用效率优化等方面。但随着算法处理数据量指数级的增长，以及大数据并行计算模型 MapReduce 的逐渐普及，算法逐渐从单机模型向分布式模型方向发展，并且算法的计算及资源利用效率等方面也逐渐受到学术界及工业界的重视。

文献[7,8]尝试借助 MapReduce 程序的并行性来提高基于用户(User-Based)的协同过滤算法的计算效率，在 Hadoop 平台上进行了算法的实现，并通过灵活设置作业 map 及 reduce 任务的数量，进一步提高了算法的计算效率。Schelter 等人[9]针对近邻(similarity-based neighborhood)协同过滤算法在面对海量数据增长时可扩展性差的问题，基于 MapReduce 模型对算法进行了重新实现，并通过 7 亿 Yahoo 音乐数据的测试，验证了算法在计算效率上的提升。文献[10]将 MinHash 聚类、概率潜在语义索引（PLSI）及 Covisitation 计数技术引入到 Google news 的协同过滤算法中，其中 MinHash 进行概率聚类适用于聚类精度要求不高的场景，在此基础上再利用 Mapreduce、Bigtable 等技术进一步提高算法的计算速度。文献[11]将协同过滤算法中最核心的计算步骤切分成四个 MapReduce 作业，并提出通过数据分区策略减小算法执行过程中的数据传输量，实验表明有效地提高了 Itembased 推荐算法的网络资源的利用效率。但不管是 UserBased 还是 ItemBased 协同过滤算法，MapReduce 环境下的协同过滤算法都需要多个作业协作完成，MapReduce 作业之间存在着大量的冗余 I/O 及资源重复申请操作，算法在资源利用效率上还存在着较大的优化空间。目前，面对 MapReduce 数据挖掘类算法资源效率低下的问题，已有工作选择对算法进行平台移植，例如将算法从 Hadoop 平台移植到 Spark 平台[12]，利用 Spark 内存及迭代计算的优势，达到提高算法效率的目的。虽然此方法能够带来不错的优化效果，但是算法平台的迁移主要存在以下两个方面的问题：

a) 迁移成本问题。将算法从已有的 Hadoop 迁移到 Spark 平台，由于开发语言从 Java 到 Scala，算法整个实现的业务代码都必须按照新的 API 及语法重写[12]。迁移过程中，研发人员面临较高的学习及开发成本，而系统管理人员面临较高的系统迁移及部署成本。

b) 稳定性问题。算法进行平台迁移后，由于新平台未经过长期的稳定性测试，新平台下的算法稳定性容易影响上层应用的服务质量。

基于以上讨论，本文与以往工作不同的是：以 ItemBased 推荐算法为例，分析其在 MapReduce 平台中存在的资源效率缺陷，提出将同一算法协同的多个 MapReduce 作业之间的 I/O 数据进行统一的缓存处理，整体上减少算法计算过程中冗余的 I/O 操作，从而达到优化资源利用效率的目的；与以往平台移植的方案相比，不需要重新开发业务代码，节省迁移及部署成本的同时，保证了系统的稳定性。

2　多 MapReduce 作业协作的资源效率缺陷

由于物品与物品之间的相似度较为稳定，所以 ItemBased 推荐算法相比其他算法(如基于流行度、基于内容、基于模型的算法等)更加适合做离线计算，应用也最为广泛。ItemBased 算法的核心思想是“物以类聚”，即假设能够引起用户兴趣的 Item 必定与其评分高的 Item 相似。算法首先计算用户对物品的喜好程度，然后根据用户的喜好计算 Item 之间的相似度，最后找出与每个 Item 最相似的 top-N 个 Item。表 1 所示为 Hadoop 平台下机器学习库 Mahout 中 ItemBased 推荐算法的输入参数。

表 1　ItemBased 推荐算法主要参数

Table 1　Main parameters of itembased recommendation algorithm

参数名	意义
input	输入 HDFS 文件地址,其中行数据文件格式为: userid itemid preference
output	算法结果 HDFS 文件系统输出路径
numRecommendations	推荐数量(Top-N)
usersFile	推荐的 userFile
itemsFile	推荐的 itemFile
maxPrefsPerUser	阈值设置：最大偏好值
minPrefsPerUser	阈值设置：最小偏好值
maxSimilaritiesPerItem	给每一个 Item 计算最多的相似 item 数目
threshold	去除低于参数 threshold 的 item 对
similarityClassname	指定进行相似性计算时调用的方法类名

ItemBased 算法主要包括以下四个阶段：a)准备 User-Item 与 Item-Item 两个矩阵；b)计算 User-Item 与 Item-Item 两个矩阵的相似度；c) partialMultiply，即将相似度矩阵用相同的 Item 作为 key 聚合到一起，为后续的矩阵乘法做准备；d) 计算推荐向量。进一步分解，可将以上四个步骤细分为以下九个 MapReduce 作业，具体的 MapReduce 作业所属阶段即执行顺序如表 2 所示。

单个 MapReduce 作业通过任务调度器分解为多个 map 及 reduce 任务，通过 input 参数的设置值，输入数据从 HDFS

等文件系统中读入后送入相应的 Map 任务。当 Map 计算结束后，数据通过 Shuffle 与 Sort 操作将<K,V>键值对数据发送到指定的 Reduce 任务，其中 Reduce 计算过程中以<key, Iterator<value>>作为数据输入格式，最后将 Reduce 结果写入到 output 参数指定的文件路径中。由于 MapReduce 作业的独立性，即任意 MapReduce 作业都需要独立的进行任务调度、资源分配、HDFS 数据读取、任务计算、数据 Shuffle、结果输出等一系列复杂的操作。特别对于 ItemBased 推荐算法包含九个 MapReduce 作业的情况，以磁盘 I/O 资源为例，算法执行过程中需要分别执行多达 18 次对 HDFS 的读取与写入操作。这些高频次的数据读写操作降低 MapReduce 集群资源的利用效率的同时，降低了算法整体的运行效率。并且由于磁盘 I/O 资源是 MapReduce 集群的性能瓶颈所在，高频次的磁盘 I/O 资源申请及释放操作容易产生资源竞争，使得 Map 与 Reduce 任务之间容易出现等待现象，不同程度上进一步降低了算法的计算效率。通过以上分析表明 MapReduce 环境下的 ItemBased 算法性能优化并不能单从优化算法本身入手，MapReduce 作业之间的独立性导致的重复磁盘 I/O 读取与写入操作，是导致 ItemBased 算法执行过程中资源利用与运行效率低下的主要原因。

表 2　ItemBased 推荐算法 MapReduce 作业执行分解

Table 2　Mapreduce job execution decomposition of itembased recommendation algorithm

步骤	MapReduce 阶段	MapReduce 作业名称
1	PreparePreferenceMatrixJob	ItemIDIndexMapper-Reducer
2		ToItemPrefsMapper-Reducer
3		ToItemVectorsMapper-Reducer
4	RowSimilarityJob	CountObservationsMapper-Reducer
5		VectorNormMapper-Reducer
6		CooccurrencesMapper-Reducer
7		UnsymmetrifyMapper-Reducer
8	partialMultiply	partialMultiply
9	RecommenderJob	PartialMultiplyMapper-Reducer

3　ItemBased 算法 MapReduce 作业资源优化

3.1　ItemBased 算法多作业运行效率分析

如表 2 所示，ItemBased 推荐算法中，一次计算过程切分为 PreparePreferenceMatrixJob、RowSimilarityJob、partialMultiply 及 RecommenderJob 四个阶段，并包括 ItemIDIndexMapper-Reducer 等在内的九个 MapReduce 作业。设 MapReduce 平台下某算法的运行总时间为 T_{sum}，计算过程被切分为 N 个 MapReduce 作业，并设每个 MapReduce 作业的完成时间为 T_i。当一个 MapReduce 作业完成后，下一 MapReduce 作业执行前需要进行资源准备，设两个 MapReduce 作业 i 与 $i+1$ 之间的资源准备时间为 P_i^{i+1}，那么算法的总体完成时间 T_{sum} 为

$$T_{sum}=\sum_{i=1}^{N}T_i+\sum_{i=1}^{N-1}P_i^{i+1} \tag{1}$$

为了排除资源状态对真实的 MapReduce 作业运行时间的影响，将受资源状态影响较大的数据输入时间 T_i^{input}，计算结果输出时间 T_i^{output} 以及作业执行过程中资源等待时间，T_i^{wait} 等可变时间称为资源准备时间。那么某 MapReduce 作业 i 的资源准备时间 P_i^{i+1} 为

$$P_i^{i+1}=T_i^{input}+T_i^{output}+T_i^{wait} \tag{2}$$

而排除资源状态对 MapReduce 作业执行效率的影响后，单个 MapReduce 作业的实际运行时间由每个作业的子阶段 map、shuffler 及 reduce 的运行时间累加而成：

$$T_i=T_{map}+T_{shuffler}+T_{reduce} \tag{3}$$

由式(1)~(3)，可以得到整个算法的执行时间 T_{sum}：

$$T_{sum}=\sum_{i=1}^{N}(T_{map}+T_{shuffler}+T_{reduce})_i+\sum_{i=1}^{N-1}(T_i^{input}+T_i^{output}+T_i^{wait}) \tag{4}$$

式(4)中，作业的子阶段 map、shuffler 及 reduce 的运行时间与任务数量、节点计算能力、资源状态及数据量等因素关系密切。但是在任务数量、数据量及节点计算能力相同的条件下，资源状态则是影响作业运行时间的主要因素。由于不同 MapReduce Job 之间的独立性，使得即使是同一算法的不同作业之间，在计算过程中，存在严重的磁盘 I/O 冗余及资源重复申请操作，导致 MapReduce 算法资源利用效率严重降低。根据式(4)，理论上提高 MapReduce 作业之间资源的利用效率(协作效率)，打破作业之间独立性的缺陷，能够达到提高 MapReduce 复杂算法计算效率的目的。

3.2　多 MapReduce 作业资源优化方法

通过第 2 章的缺陷分析及 3.1 节的理论分析，表明提高 MapReduce 作业之间资源的协作效率，解决作业之间独立性带来的资源重复操作，能够提高作业对的资源利用效率。总体上，可以通过以下两种方法来优化多 MapReduce 作业之间的资源效率：a)利用 MapReduce 框架内置的分布式缓存(DistributedCache)机制，打破 MapReduce 作业之间资源共享的屏障。缓存系统能够将共享数据分发到各计算节点的内存中，提高作业的计算效率；b)在基于磁盘的分布式文件(如 HDFS)之上，引入以内存为中心的虚拟的分布式存储系统，将多 MapReduce 作业之间共享的数据资源从 HDFS 中加载到分布式共享内存中，从而减少对磁盘 I/O 的调用次数。优化方法 a)的优点是针对特定算法级别的，灵活性较高；但是需要对算法部分代码进行重构和部署，需要对算法代码进行部分修改。优化方法 b)将新的分布式内存文件系统部署到生产系统中，容易对整个系统的稳定性造成影响，但是省去了对算法代码的修改。针对单个的 ItemBased 算法，本文中采用方法 a)来提高资源的利用效率。如下算法所示为基于 DistributedCache 的 ItemBased 算法步骤。

算法 1　基于 DistributedCache 的 ItemBased 算法.

INPUT:Parameter1：<userID，itemID，score>用户评分矩阵，Parameter2:*InputURL*.

OUTPUT:Parameter1：<userID，itemIDS>推荐结果，Parameter2: *OutputURL*.

① **for** *i*=0 to *readData*(*InputURL*).*blocksize-1* **do**

②　block ← MEMORY.*load*(*InputURL*)//读取文件块

③　DistributedCache.*addCacheFile*(block)//将文件块加载内存

④ **end for**

⑤ **foreach** job **in** List<PreparePreferenceMatrixJob> *list*

⑥　MapReduce.*start*().*getJob*(*list*)

⑦　context.*setCacheFiles*(PreparePreferenceMatrixJob)

⑧　context.*getCacheFiles*()

⑨ **end for**

⑩ **foreach** job **in** List< RowSimilarityJob> *list*

⑪　MapReduce.*start*().*getJob*(*list*)

⑫　context.*setCacheFiles*(RowSimilarityJob)

⑬　context.*getCacheFiles*()

⑭ **end for**

⑮ MapReduce.*start*().*getJob*(partialMultiply)

⑯ context.*setCacheFiles*(partialMultiply)
⑰ context.*getCacheFiles*()
⑱ context.*setCacheFiles*(PartialMultiplyMapper-Reducer)
⑲ context.*getCacheFiles*()
⑳ MapReduce.*start*().*getJob*(PartialMultiplyMapper-Reducer)

㉑ **return** Job.*addCacheFile*(new File("*OutputURL*"))

算法①~④行将 ItemBased 算法根据输入参数 *InputURL* 将输入数据加载到分布式缓存中；⑤~⑨行执行 PreparePreferenceMatrixJob 阶段所有的 MapReduce 任务，使得 ItemIDIndexMapper-Reducer、ToItemPrefsMapper-Reducer 及 ToItemVectorsMapper-Reducer 三个 Job 之间在内存中实现了数据共享；⑩~⑭行执行 RowSimilarityJob 阶段所有的 MapReduce 任务，DistributedCache 实现了 Count-ObservationsMapper-Reducer、VectorNormMapper-Reducer、CooccurrencesMapper-Reducer 及 UnsymmetrifyMapper-Reducer 四个 Job 之间的内存数据共享；由于 partialMultiply 与 RecommenderJob 两个阶段由单个 MapReduce 作业组成，⑮~⑳实现了 partialMultiply 与 PartialMultiplyMapperReducer 两个作业的内存数据读写操作；最后，㉑行将算法计算结果写入内存级的文件中。

4　实验评价与比较

4.1　实验环境配置

为了对比 ItemBased 算法在不同测试环境下的执行效率及资源利用情况，本文设计了两组实验：实验 1 为 MapReduce 原生环境及 DistributedCache 内存缓存条件下数据吞吐量的测试；实验 2 为 ItemBased 推荐算法在 MapReduce 原生环境及 DistributedCache 内存缓存条件下的对比测试。实验集群节点总数为 11，其中管理(Master)节点数为 1，作业执行节点(Worker)数为 10。集群节点实验环境如表 3 所示。

表 3　总体实验环境描述

Table 3　Description of experimental environment

项目	描述
操作系统	Ubuntu 14.04.5 LTS (Trusty Tahr)
JVM Version	1.8 for Linux OS
Hadoop Version	2.7.1
节点 CUP	Intel core i5 Skylake 3.2GHz
节点内存	8GB-DDR4- 2400MHz (4GB*2)
节点硬盘	Seagate Barracuda 1TB 7200 32MB SATA3 ST31000524AS
网卡信息	TP-LINK TG-3269C PCI RJ-45

4.2　实验 1　MapReduce 作业数据吞吐量对比测试

为了测试 MapReduce 原生环境及 DistributedCache 内存缓存条件下数据吞吐量的区别，本实验协同执行 10 个读取文本文件的 MapReduce Job，这样可以将实验的 MapReduce 作业的性能瓶颈控制到 I/O 性能上。实验数据量大小为 40 GB 的文本文件，HDFS 及 DistributedCache 中数据块大小分别设置为 512 MB 与 1 GB 不同的两组。HDFS 中数据块的分布策略为默认的机架感知模式，每组分别进行三次测试，首先当数据块大小为 512 MB 时测试结果如表 4 所示。

将 HDFS 中的数据块大小配置项 dfs.block.size 以及 DistributedCache 中的数据块大小配置项从 512 MB 设置为 1 GB，再次进行三次测试，得到测试结果如表 5 所示。

表 4　数据块大小为 512 MB 时的测试结果

Table 4　Test results when data block size is 512 MB

编号	耗时/s	HDFS	DistributedCache	性能提升
1	Application total time	238	203	14.71%
	Job total time	227	188	17.18%
	Average map running time	8	6	25%
	Shuffler running time	177	153	13.56%
2	Application total time	245	203	17.14%
	Job total time	226	186	17.7%
	Average map running time	8	6	25%
	Shuffler running time	178	153	14.04%
3	Application total time	244	202	17.21%
	Job total time	228	188	17.54%
	Average map running time	8	6	25%
	Shuffler running time	177	151	14.69%

表 5　数据块大小为 1 GB 时的测试结果

Table 5　Test results when data block size is 1 GB

编号	耗时/s	HDFS	DistributedCache	性能提升
1	Application total time	204	141	30.88%
	Job total time	176	120	31.82%
	Average map running time	15	9	40%
	Shuffler running time	132	87	34.09%
2	Application total time	187	131	29.95%
	Job total time	169	116	31.36%
	Average map running time	15	10	33.33%
	Shuffler running time	128	86	32.81%
3	Application total time	187	136	27.27%
	Job total time	174	118	32.18%
	Average map running time	15	10	33.33%
	Shuffler running time	133	88	33.83%

根据表 4 与 5 中的数据可知，相比原生态的 HDFS，无论是比较应用总体完成时间、作业总体完成时间、Map 任务平均完成时间以及 Shuffler 运行时间，DistributedCache 均能够提高 MapReduce 作业读取数据的速度。特别的，对比表 4 与 5 的数据，可以发现数据块大小为 1 GB 时效率比 512 MB 效率提升更大。这是由于当数据块变大时，在作业数据集大小固定的前提下，整个 MapReduce 作业 Map 任务数量减少。由于相同集群中资源数量固定，所以资源竞争压力小，导致 Map 任务的读取速度变大，从而出现数据块越大，性能提升越明显的现象。

4.3　实验 2　ItemBased 推荐算法对比测试

实验中所采用的 ItemBased 推荐算法的实现版本来自于 mahout 的 0.11.1 版本，具体的算法运行类入口为 org.apache.mahout.cf.taste.hadoop.item 包下的 RecommenderJob 类，使用原生 HDFS 文件系统为存储目标时的运行命令为：

```
./hadoop jar
/home/hadoop/mahout0.11.1/mahout-examples-0.11.1-job.jar
org.apache.mahout.cf.taste.hadoop.item.RecommenderJob
-i /mahout/itemcf/inputdata
-o   /mahout/itemcf/result
-s SIMILARITY_LOGLIKELIHOOD
--tempDir /mahout/itemcf/temp1
```

其中：inputdata 为输入数据目录；result 为输出文件目录；temp 为作业运行过程中的临时文件目录。当使用 DistributedCache

为存储系统时，按照算法 1 对 ItemBased 算法进行重构。重构后，由于需要在 DistributedCache 中运行，其运行命令与 HDFS 不同，其运行命令为：

```
./hadoop jar
/home/hadoop/mahout0.11.1/mahout-examples-0.11.1-job.jar
org.apache.mahout.cf.taste.hadoop.item.RecommenderJob
-i   distributedCache://ubuntu201:23456/ratings/ratings.data
-o   distributedCache://ubuntu201: 23456/ratings/result
-s   SIMILARITY_LOGLIKELIHOOD
--tempDir distributedCache://ubuntu201:19998/ratings/temp
```

其中：distributedCache://ubuntu201:19998/ ratings/ratings.data 为算法输入数据路径；distributedCache://ubuntu201:23456/ ratings/result 为输出数据路径；distributedCache://ubuntu201: 23456/ratings/temp 为临时文件目录；SIMILARITY_ LOGLIKELIHOOD 为相似性计算参数。

实验中，用户评分矩阵中<userID, itemID, score>数据对为 2 000 万，将实验运行三次取平均值，得到 HDFS 与 DistributedCache 资源效率对比如表 6 所示。

据表 6 中实验数据，当存储系统为 HDFS 时，算法总的资源等待时延为 181 s；当存储系统为 DistributedCache 时，算法总的资源等待时延缩短至 39 s。利用 DistributedCache 重构后的算法，大大提高了作业的资源利用效率，资源等待总时延从原来的 181 s 减少为 39 s，资源效率提高了 364.1%。实验结果正好验证了 3.1 节中对 MapReduce 作业运行效率的分析，证明 DistributedCache 对资源的准备时间，即资源的利用效率方面的极大改进。

表 6　计算数据量为 2 000 万时计算效率对比

Table 6　Comparison of computational efficiency when data is 20 million

ItemBased 算法阶段切分			运行时间/s			
Step	MapReduce 阶段	MapReduce 作业名称	HDFS 作业运行时间	HDFS 资源等待时延	DistributedCache 作业运行时间	DistributedCache 资源等待时延
1	PreparePreferenceMatrixJob	ItemIDIndexMapper-Reducer	201	64	199	8
2		ToItemPrefsMapper-Reducer	269	15	268	4
3		ToItemVectorsMapper-Reducer	207	14	207	3
4	RowSimilarityJob	CountObservationsMapper-Reducer	195	12	192	3
5		VectorNormMapper-Reducer	218	16	216	4
6		CooccurrencesMapper-Reducer	436	14	433	4
7		UnsymmetrifyMapper-Reducer	423	13	424	4
8	partialMultiply	partialMultiply	222	17	218	5
9	RecommenderJob	PartialMultiplyMapper-Reducer	826	16	821	4

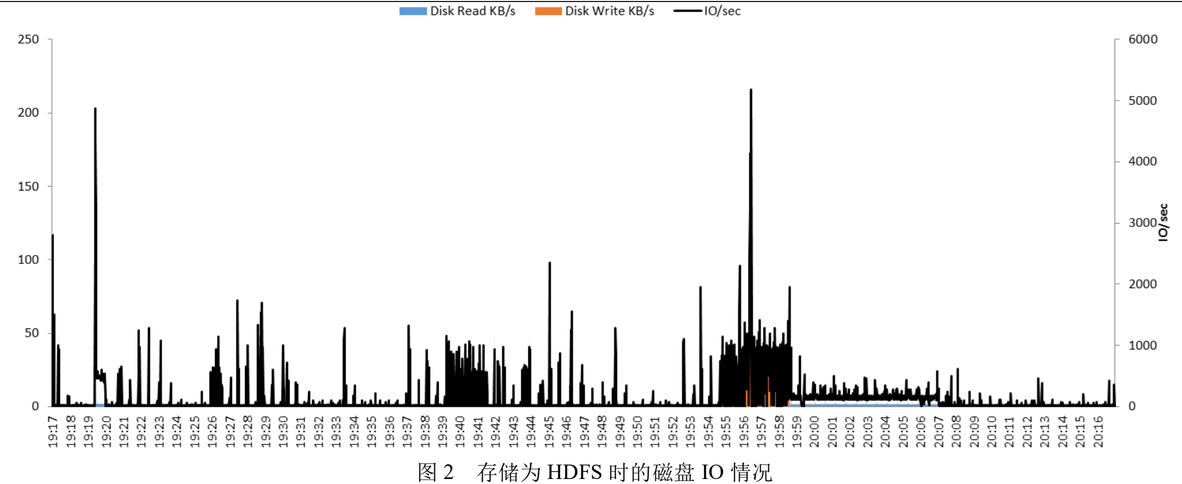


图 2　存储为 HDFS 时的磁盘 IO 情况

Fig. 2　Disk IO situation when storing as HDFS

图 3　存储为 DistributedCache 时的磁盘 IO 情况

Fig. 3　Disk IO situation when storing as distributedcache

算法在不同的条件下，运行过程中的对磁盘 I/O 使用情况监控如图 2 与 3 所示。其中图 2 为存储系统为 HDFS 时的磁盘 I/O 使用情况，图 3 为存储系统为 DistributedCache 时的磁盘 I/O 使用情况。比如两图可以发现相比原生的 HDFS，DistributedCache 的磁盘 I/O 在作业运行过程中几乎为空闲状态，只有在最后作业运行完毕，即将作业运行结果写入 HDFS 时，才产生了较高的磁盘 I/O 资源消耗，这也进一步验证了实验结果中作业资源准备时间大大缩短的原因。使用 DistributedCache 时，快速的内存 I/O 代替了密集的磁盘 I/O，不仅优化了作业运行过程中的资源利用效率，大大缩短了资源的准备时间，从而整体上提高了作业的运行效率。

5 结束语

MapReduce 计算模式逐渐成为并行计算标准的同时，也存在一定的资源利用效率问题；特别在需要多个 MapReduce 作业协作完成的复杂大数据挖掘类算法场景，多个作业之间严重的冗余磁盘读写及重复的资源申请操作，使得 MapReduce 算法的执行时间、资源利用、能耗等方面的效率较低。本文发现造成该问题的原因是由于 MapReduce Job 的独立性，使得同一算法的不同作业之间存在严重的磁盘 I/O 冗余及资源重复申请操作，从而导致 MapReduce 算法执行效率严重降低。所以，本文首先对 MapReduce 环境下的 ItemBased 算法资源效率缺陷进行了分析，发现存在问题的同时提出利用 DistributedCache 缓存计算过程中产生的中间数据，从而减少算法执行过程中冗余的 I/O 操作，达到优化资源利用效率的目的。最后，通过两组实验证明了使用 DistributedCache 时，快速的内存 I/O 代替了密集的磁盘 I/O，不仅优化了作业运行过程中的资源利用效率，大大缩短了资源的准备时间，从而整体上提高了资源的利用效率。通过实验结果表明，优化后的 I/O 资源效率提高 3 倍以上。下一步工作主要研究在集群负载压力较大时，即集群中同时运行多组大数据挖掘类算法时，DistributedCache 资源的使用及调度效率。

参考文献：

[1] The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east [EB/OL]. [2018-3-15]. http://www. emc. com/collateral/analyst-reports/idc-the-digitaluniverse-in-2020. pdf.

[2] Ghemawat S, Gobioff H, Leung S T. The google file system [C]// Proc of the 19th ACM Symposium on Operating System Principles. New York: ACM Press, 2003: 29-43.

[3] Chen C, Lin J, Kuo S. MapReduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems [J]. IEEE Trans on Cloud Computing, 2018, 6 (1): 127-140.

[4] 廖彬, 张陶, 于炯, 等. MapReduce 能耗建模及优化分析 [J]. 计算机研究与发展, 2016, 53 (9): 2107-2131. (Liao Bin, Zhang Tao, Yu Jiong, *et al.* Energy consumption modeling and optimization analysis for MapReduce [J]. Journal of Computer Research and Development, 2016, 53 (9): 2107-2131.)

[5] 吴倩, 王林平, 罗相洲, 等. 基于 MapReduce 的 top-k 高效用模式挖掘算法 [J]. 计算机应用研究, 2017, 34 (10): 2897-2900. (Wu Qian, Wang Linping, Luo Xiangzhou, *et al.* Top-k high utility pattern mining algorithm based on MapReduce [J]. Application Research of Computers, 2017, 34 (10): 2897-2900.)

[6] 廖彬, 张陶, 于炯, 等. 温度感知的 MapReduce 节能任务调度策略 [J]. 通信学报, 2016, 37 (1): 61-75. (Liao Bin, Zhang Tao, Yu Jiong, *et al.* Temperature aware energy-efficient task scheduling strategies for mapreduce [J]. Journal on Communications, 2016, 37 (1): 61-75.)

[7] Zhao Zhidan, Shang Mingsheng. User-based collaborative-filtering recommendation algorithms on Hadoop [C]// Proc of International Conference on Knowledge Discovery and Data Mining. Piscataway, NJ: IEEE Press, 2010: 478-481.

[8] Ma M M, Wang S P. Research of user-based collaborative filtering recommendation algorithm based on Hadoop [C]// Proc of International Conference on Computer Information Systems and Industrial Applications. New York: Atlantis, 2015: 63-66.

[9] Schelter S, Boden C, Markl V. Scalable similarity-based neighborhood methods with MapReduce [C]// Proc of ACM Conference on Recommender Systems. New York: ACM Press, 2012: 163-170.

[10] Das A S, Datar M, Garg A, *et al.* Google news personalization: scalable online collaborative filtering [C]// Proc of International Conference on World Wide Web. New York: ACM Press, 2007: 271-280.

[11] Jiang J, Lu J, Zhang G, *et al.* Scaling-Up Item-Based Collaborative Filtering Recommendation Algorithm Based on Hadoop [C]// Proc of IEEE World Congress on Services. Piscataway, NJ: IEEE Press, 2011: 490-497.

[12] 廖彬, 张陶, 国冰磊,等. 基于 Spark 的 ItemBased 推荐算法性能优化 [J]. 计算机应用, 2017, 37 (7): 1900-1905. (Liao Bin, Zhang Tao, Guo Binglei, *et al.* Performance optimization of ItemBased recommendation algorithm based on Spark [J]. Journal of Computer Applications, 2017, 37 (7): 1900-1905.)